Neil Brown, the maintainer of the Linux NFS client, uses SpecSFS'97 to test NFS performance, and so do most people.
But 1997 was a long time ago... Does the SpecSFS '97 accurately reflect the way that NFS is used today?

Improving NFS performance

Work In Progress

Shehjar Tikoo and Peter Chubb

Gelato Project

National ICT Australia and The University of New South Wales

April 2006

NFS too slow?

- Many reports of slowness
- No real data

This talk

- Work in progress Incomplete

  Attempt to:

  – validate current benchmarks

  – Find and fix bottlenecks

  – and maybe ... verify reports of slowness

So what do you do if you have no data? Try to measure.

Measurements

- Standard filesystem benchmarks

  – Reflect single-client load (e.g., Bonnie, iozone)

  – Multiple client machines perturb file access patterns

  – Nice for quick'n'dirty evaluation

  – Nice for whole-system (client plus server) evaluation

You can use standard filesystem benchmarks such as iozone, but these measure too much: I want to be able to split client and server issues as far as possible. For initial 'is there a problem' benchmarks they can be very valuable, however.

Measurements

- Network filesystem benchmarks
  - Old (NFSstone)
  - Old (SpecSFS ’97) and proprietary

The Network file system benchmarks are either old (NFSstone) or both old and proprietary (SFS). That being so, SFS ’97 version 3.0 (SFS3.0) is still the most commonly used NFS benchmark.

Query:

Does SpecSFS ’97 represent today’s workloads?

Given that SpecSFS ’97 version 3.0 was based on workloads measured in ’96–’97 on NFS version 2, and extrapolated to version 3, is it still the most useful tool?

SpecSFS ’97

- Generates load according to preset pattern
- Parameterisable
- SPECified benchmark set of reportable parameters

SPEC always gives very tight specifications of a ‘reportable’ benchmark run. The SFS code itself can be tailored to reproduce many different loads; but it comes pre-set-up for the reportable load.
The reportable load was derived by taking measurements on a large number (> 1000) of different systems, then doing statistical analysis to find a common load. The resulting operation mix was then extrapolated to NFS version 3, and verified on a few servers at Sun Microsystems.

Changes since ’97

Attribute	1997	2006
Network	10BMb/s	1Gb/s
Client Memory	16MB	1 GB
Server Memory	1GB	4–16G
NFS version	2	3
Average file size	??	??

Lots of things have changed in ten years. In 1997, 10-base-T was the most common interconnect — 100MB/s gear was available, but expensive — in 2006, 100MB/s is common, and new equipment comes standard with 1000Mb. In 1997, I upgraded one of my servers from 1M of memory to 4M, and thought I was getting a good deal. In 2006, a server with less than a gigabyte of memory isn’t worthy of the name, and even desktops tend to have more than 512M. Disk sizes have also increased massively.
Have file sizes increased as well? See below...

What we collect

- timestamp, anonymised request and reply parameters,
- For 24 hours (around 8G data)

What we do NOT collect

- real user names, ids
- file names
- IP addresses

These are replaced with

cookies

We're trying to collect anonymised traces from as many places we can, using *nfsdump*.
The nfsdump tool from Daniel Ellard (Harvard) is a modification of tcpdump; it collects NFS traffic by monitoring a network interface. It has to be run either on the server, or on another machine connected via port replicators (so it can see the same traffic that the server sees).
The raw data is very large — a 24 hour run can be hundreds of Gigabytes. After it has been anonymised, it shrinks somewhat, so the largest trace we've collected so far (for the OSC 'b' server) is only 31Gbytes.

Our Data

24 hour traces from

- UNSW CSE
- Ohio Supercomputer Center

Courtesy of Neil Brown (UNSW) and Doug Johnson (OSC), 24-hours' worth of traces were collected from eight servers — seven at OSC and one at UNSW. We'd like more traces, but haven't yet found anyone willing to get them for us.

The CSE trace

- From UNSW School of Computer Science and Engineering
- Large number (> 300) of desktop clients, most with < 1G memory
- 'Academic' workload (whatever that means)
- Out of session $\rightarrow$ no undergraduates
- Server and most clients running Linux 2.6.x
- Home directories, and /usr/local

OSC traces

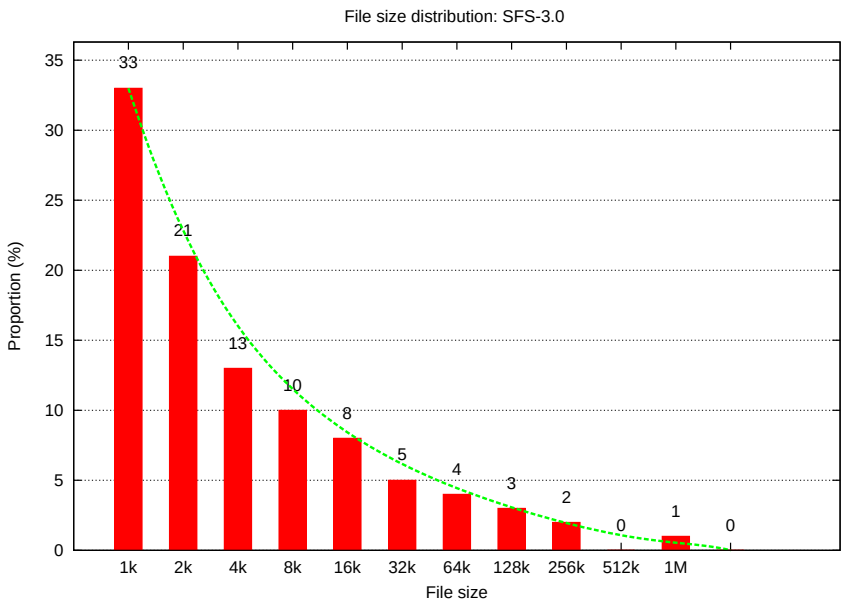
- Around 600 clients, each with around 4G memory
- Some clients grouped into clusters (e.g, ˜255-node dual I2 or xeon)
- Gigabit ethernet interconnect
- Server running Linux 2.6.7
- Home directories; 'HPC' workload.

The UNSW CSE trace covers twenty-four hours of low activity. It was taken when the University was not in session, so there were few or no undergraduates about. Most of the traffic, therefore, is for routine system administration, and postgraduate and staff work. The set up is that almost all the teaching laboratory machines and the workstations of many of the academics and postgrads, mount home directories and `/usr/local` from the server being traced. `/usr/local` contains locally modified programs and specially licensed programs. It is read-only. Home directories are of course mounted read-write.

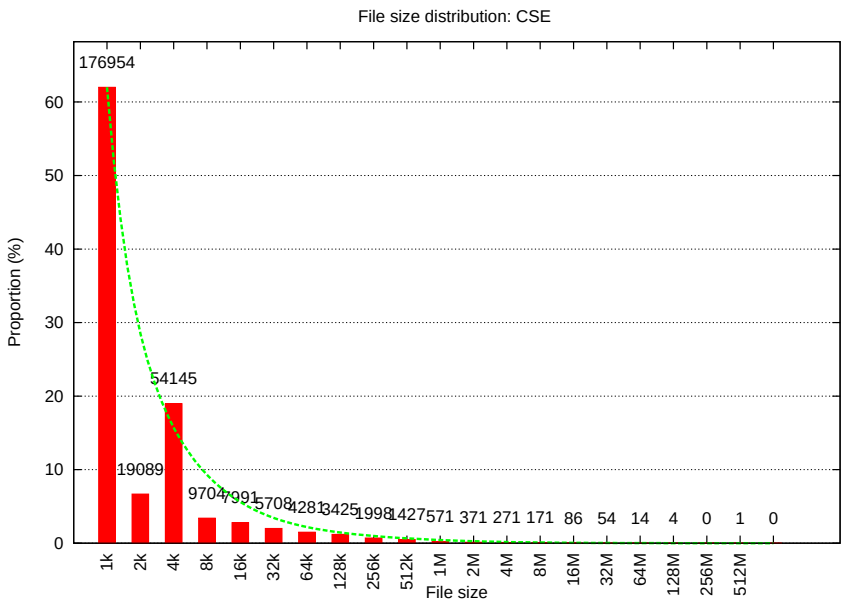
The Ohio Supercomputer Center runs seven NFS servers, each running with 4G memory, Linux 2.6.7. The clients are as follows:

Number	Type	Amount memory
255	dual cpu Itanium	4GB
258	dual cpu xeon	4GB memory
1	dual cpu xeon	2GB
5	dual cpu Itaniums	12GB
3	16 cpu altix	32GB
1	32 cpu altix	64GB
72	dual cpu Opteron	4GB
10	dual cpu Opteron	8GB
2	dual cpu Opteron	16GB

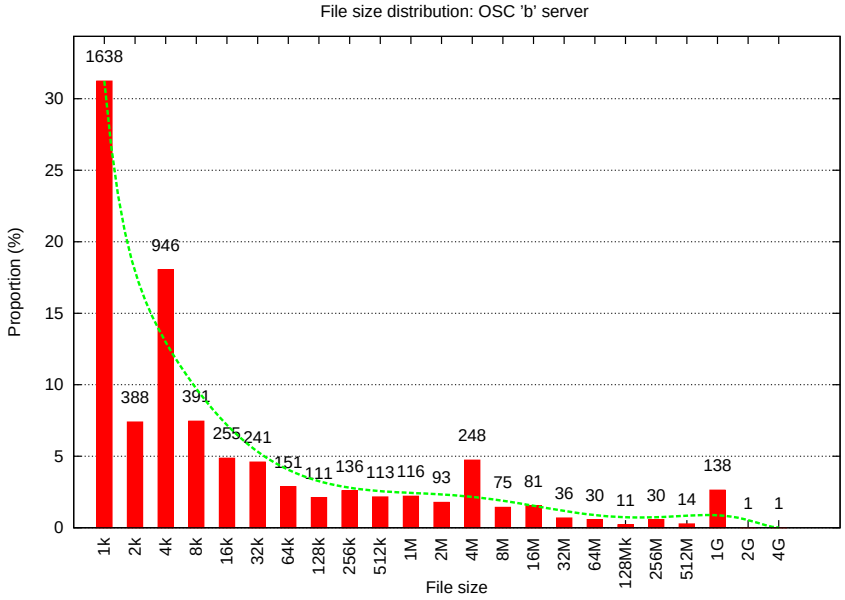
During the run, some users were running large HPC jobs on the clusters; others were doing editing/compiling etc.



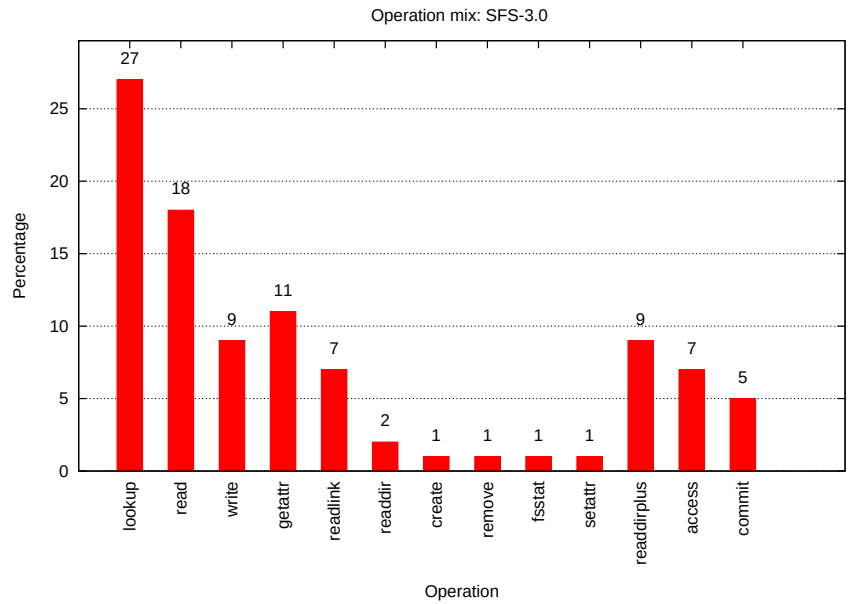
The file sizes specified for a reportable SpecSFS '97 run are from a long-tailed distribution. The largest file used is a Megabyte.



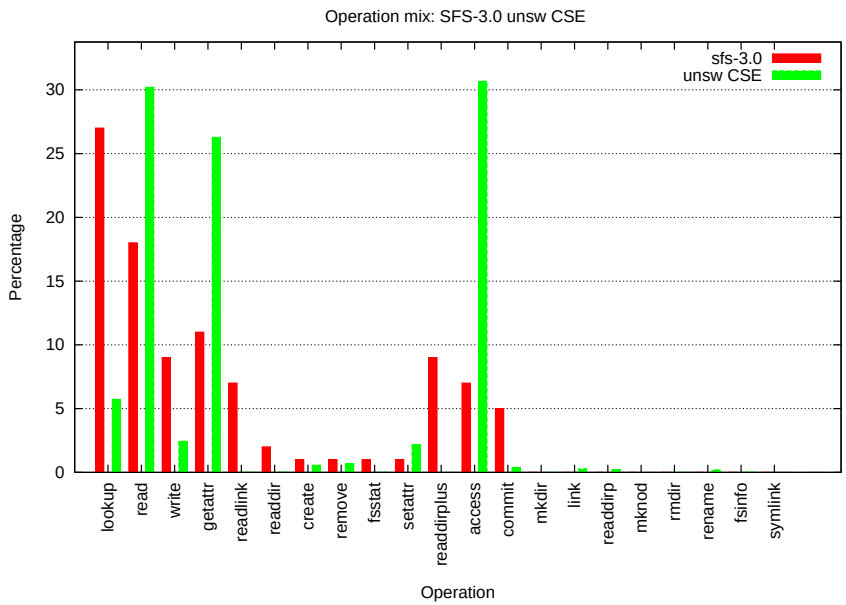
The CSE traces also have a long tail: there are significant numbers of files larger than 2G. Moreover, there are a very large number of 0-length and 1-length files, presumably used as timestamps. (These inflate the 0–1k bar). The y-axis is percentage; the numbers above each column are the actual counts.



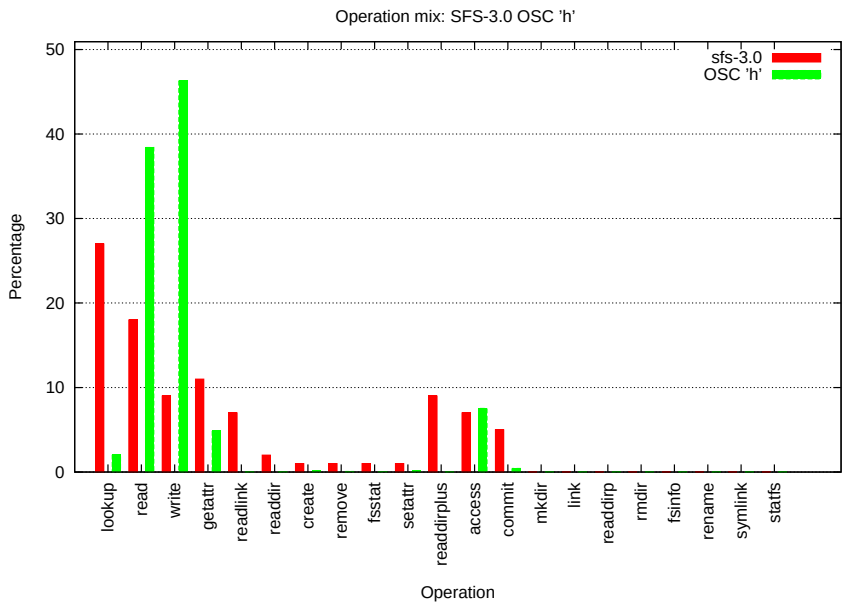
A trace obtained by Doug Johnson at the Ohio Supercomputer Center (OSC) is even more extreme. There are significant numbers of files larger than 4M, and even at 4G. Some of the other traces from the OSC show even bigger files. Again, the y-axis is percentage; the numbers above each column are the actual counts.



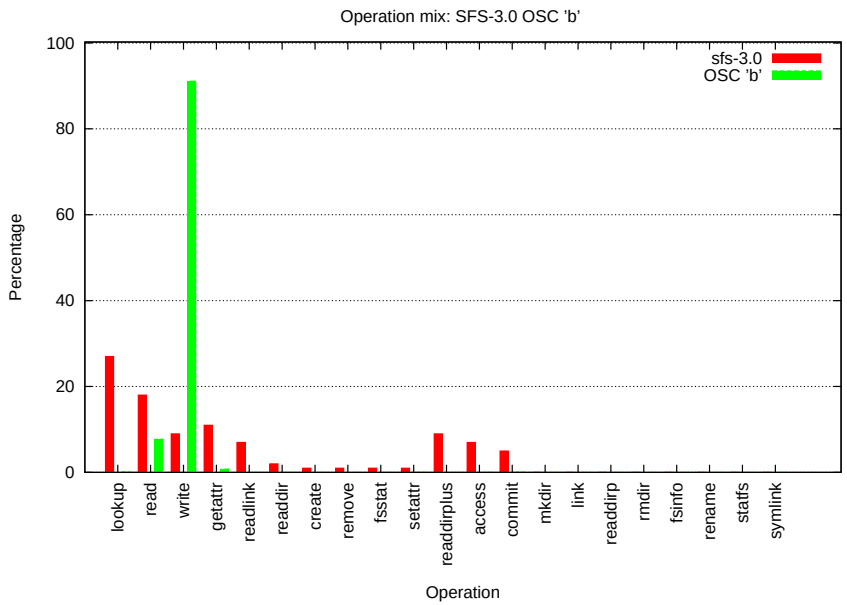
Reportable SpecSFS ’97 version 3.0 results have to be generated according to a standard proportion of each operation. Lookups, and reads dominate; also readdirplus and getattr take a significant part.



But the trace derived from UNSW CSE doesn't show this pattern at all! There are almost no lookups; the top three operations are read, getattr and access. My conjecture is that people are running `make` and `find` a lot — especially as the `GETATTR` operations are predominantly on the zero-length files.



The 'h' server from OSC shows yet another pattern. Read and write dominate; and there are slightly more writes than reads. This is surprising; conventional wisdom says that reads almost always outnumber writes by a large proportion.



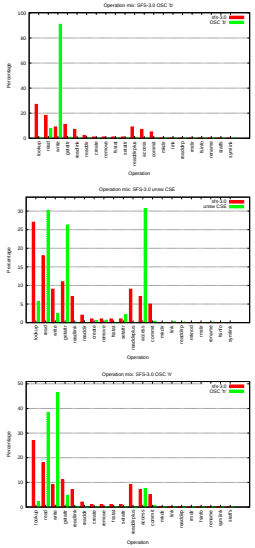
The 'b' server shows an even more surprising result: more than 90% writes! And the reads shown are largely from the *same files as the writes*. Moreover these are the few largest files that appear in the trace.

Observed patterns

Three major patterns:

1. Write-intensive
2. Getattr-intensive
3. Balanced load

None represent the SFS 3.0 load!



So what we see are three different kinds of workloads in the 8 traces gathered so far. None of them match SFS3.0; but as yet we have too few traces to draw any firm conclusions.
What is certain, is that individual workloads vary quite a bit, and it's advisable to test performance on your own workload.

Why the differences?

- *Conjecture:* Caching
  - Few lookups → Directory names cached on client
- *Conjecture:* Faster interconnect
  - NFS becomes more attractive for large files
- *Conjecture:* Delayed Commit
  - NFS more attractive for write-intensive loads
- *Conjecture:* Stupidity
  - Non-computer-science user base using NFS instead of local storage

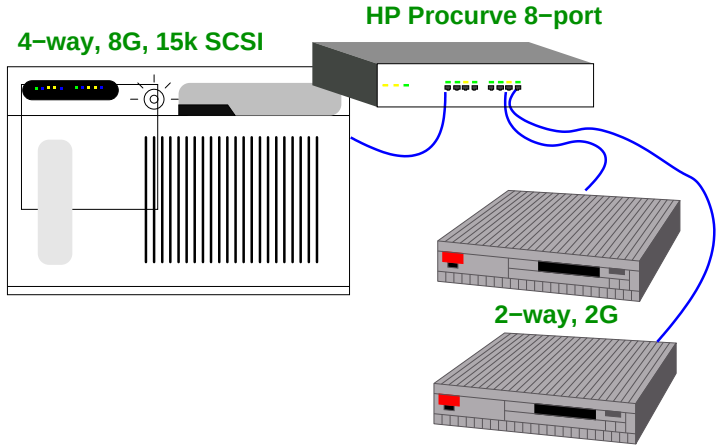
I'm conjecturing that increased memory on client and server machines is the single biggest reason for the lack of `lookup` operations. Name to inode translations have always been a potential bottleneck; current operating systems have caches of various kinds to avoid performing them as far as possible.
In addition the faster interconnect means that it's faster in many instances to get a file contents out of a server's memory than to read it off local disk.
And with delayed commit, NFS3 allows similar performance gain for writes.

Experiments

- Trace replay — later
- Simple throughput

There are two sets of experiments we want to do. Ultimately we want to be at the point where replaying arbitrary traces is easy, so that reported problems can be explored quickly. The aim is to replay a trace, but faster (possibly using multiple network links to avoid saturating the network), until the server becomes a bottleneck. Then by careful profiling on the server, the problem can be identified.
However, at present, the tools we obtained from Harvard crash on our traces. Until this can be fixed (and we're working on it) we cannot easily replay traces.

Test Bed



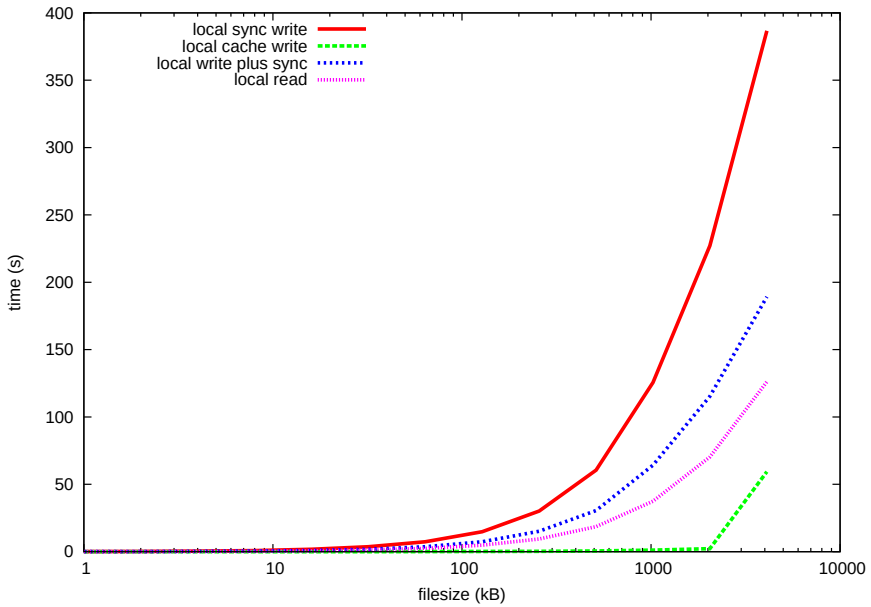
Sustained Read/Write

- Measure elapsed, real, system time on server and client
- Read/write different-sized files
- Try different parameters: tcp/udp, `O_SYNC`, etc
- Different underlying filesystems: XFS, ext[23], reiserFS

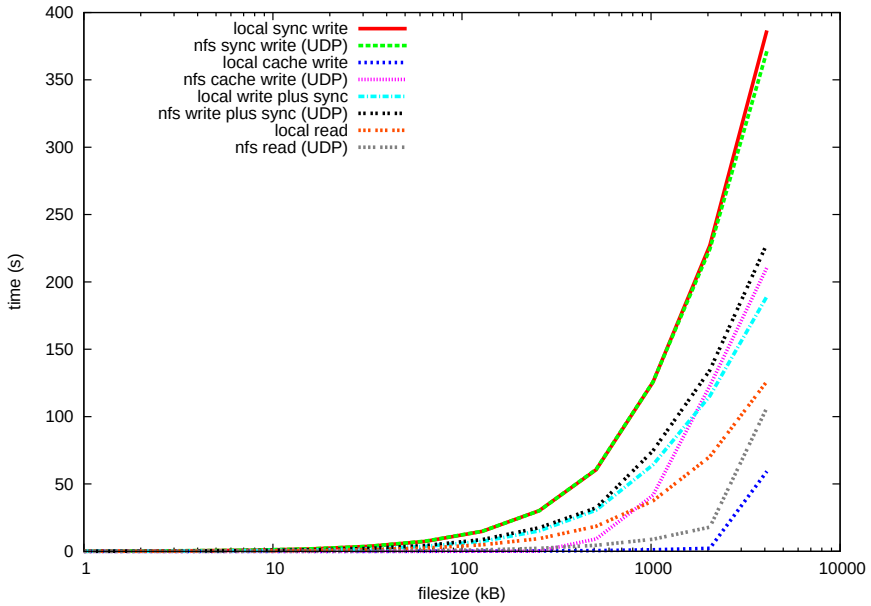
The test-bed we used consisted of an HP Procurve 8-port gigabit unmanaged switch (which we chose for its low latency — around 4 ns), some number of load-generating client machines (currently either up to 2 HP zx6000, with 2 Madison processors and 2G memory, or up to 7 Celeron procesors with 1G memory; all with gigabit ethernet); and a server. The server is either an Altix 350 with 2 nodes, 2 processors per node, and 8G memory split across the two nodes, or another 2-way zx6000.
We used the Altix for two reasons:

1. It had more memory than any other machine we have
2. It is often sold as an NFS server

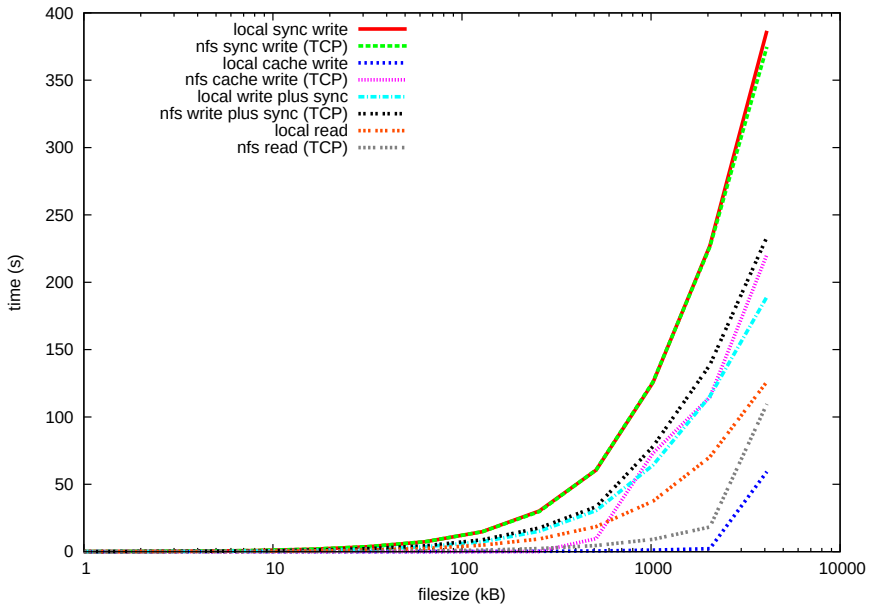
Given that streaming (or *almost* streaming) read and write loads are important (and a prime source of complaints of slowness on the Linux Kernel Mailing List), we decided to test them first (it's also easier to do than some of the other things)



First we timed local operations on the server. This is on ReiserFS version 3 — as you can see, operations on large files are relatively slow.



Using NFS over UDP, the synchronous write load shows a slight speedup for large files. I believe this is because the data can be transferred directly from the network buffer to the disk, without going through the CPU cache on the server, but haven't yet been able to confirm this.
Reads are significantly faster. In each case, the filesystem was unmounted and remounted between tests; so the local read is off the disk, but the remote read is from the server's cache.



NFS over tcp results are very similar to the UDP results.

Some Observations

- Streaming I/O seems to perform adequately.
  - Amount of memory on server and client for caching crucial for good performance.
- As clients are added, aggregate performance falls
- Because server sees almost random activity

So everything looks good so far... but we haven't yet tried replaying real loads.

Tentative Conclusions

- SpecSFS '97 version 3.0 does **not** represent today's workloads
- Single client streaming read/write performance doesn't look too bad
- Improved cache coherence protocols could make a **big** difference
  - Maybe NFS 4 will show improvement here.

Given the large number of `GETATTR` calls, something that allows clients to know that a file hasn't changed could reduce a lot of latency. NFS version 4 has the notion of a *file lease*, that says that a file is (at least temporarily) *owned* by an individual client. If another client wants to modify the file, the lease has to be broken. Using mechanisms like this, I expect that in NFS version 4 traces `GETATTR` will almost disappear (rather as `LOOKUP` has in the NFS 3 traces).

Conjecture

An Itanium makes a **great** NFS server

- Gigabit ethernet (or 10G) faster than single spindle local disc
- Huge address space enables massive server page cache
- Leverage sharing across client network

My conjecture is that Itanium machines could be configured as really good NFS servers. You don't need many processors, just huge amounts of memory, network and disk. The large address space, high memory bandwidth, high I/O bandwidth and good manageability of Itanium machines make them almost ideal.

Where to from here?

- Start replaying real traces (to find bottlenecks)
- Get more traces

  You can help

  http://www.gelato.unsw.edu.au/IA64wiki/NFSBenchmarking

- Use Markov techniques to synthesize similar traces
- Analyse server and client behaviours under load.

Replaying traces should start happening soon.
But we really need more traces. We have too few to make any kind of categorical statement that the Spec SFS workload is non-representative. You can help here, if you run an NFS server, and can capture some traffic and ship us the anonymised results...
But the traces are huge; one thing we're thinking about is to analyse the traces on-site, then generate a set of Markov matrices that can be shipped to us for analysis and reply. (A Markov matrix is a set of probabilities, that given the last event was x that the next event is y. For NFS the *event*s would be (file, operation) pairs; we're still working out exactly what form they should take, and what the minimum information we need is to be able to recreate traces of the same character as those observed.
Markov matrices have two major advantages:

1. They are a lot easier to check (to make sure no sensitive information is shipped to us)
2. They are a lot smaller than the corresponding traces (a few kB as opposed to tens of GB).

Acknowledgements

Neil Brown (UNSW) Provided CSE traces

Doug Johnson (Ohio Supercomputing Center) Provided OSC traces

SGI provided an Altix machine

HP provided servers and clients

Daniel Ellard (Harvard) provided trace capture/replay tools

This work was funded by the Australian Research Council, HP, UNSW, and National ICT Australia.

Nothing we do is on our own; thanks to everyone who helped.